MAX+PLUS II Tutorial Manual

Version 2002

for the Computer Engineering course
ECE-4DM4 - Computer Architecture (Term 1)

Version 2002 Prepared by:

Prof. Ted Szymanski
Dept. ECE, McMaster University

September 2002

Class Website:
http://www.ece.mcmaster.ca/faculty/teds/courses

Acknowledgements :

Software and hardware donations from
the *"Altera University Program"* are gratefully acknowledged.

Table of Contents

1. Introduction
1.1 Running MAX+PLUS II 2
1.2 Overview of MAX+PLUS II 3

2. Creating a design in VHDL
2.1 Example 1: A latch array 3
2.2 Entering VHDL code 4
2.3 Checking your VHDL code 4

3. Compiling your VHDL code
3.1 Brief description of compiler stages 5
3.2 Setting compiler options 6
3.2.1 Selecting a device family 7
3.2.2 Selecting a global project logic synthesis style 7
3.2.3 Turning on the smart recompile command 8
3.2.4 Enabling compilation for VHDL-93 8
3.3 Running the compiler 8

4. MAX+PLUS II Simulator
4.1 Function simulation versus Timing simulation 9
4.2 Timing simulation using a simulator channel file (.SCF) 10
4.2.1 Editing a SCF 11
4.3 Timing simulation using a vector file (.VEC) 12

5. MAX+PLUS II Timing Analyzer
5.1 Registered performance 13
5.2 Determining the critical path 14
5.3 Using the Delay Matrix 15

6. Analyzing synthesis results (.RPT file) 17

7. Changing compiler settings
7.1 Viewing logic synthesis equations 22
7.2 Optimizing logic synthesis for speed 23

8. Guidelines 23

9. Resources 26

Appendix A VHDL Description 26
Appendix B Vector file: example1.vec 28
Appendix C Our conventions on writing VHDL documentation 29

1. Introduction

The following 4DM4 user manual should get you fully operational with the MAX+PLUS II Design System.

For a more detailed manual, see our class web site (http://sparky.mcmaster.ca/teds)

1.1 Running MAX+PLUS II

- Type maxplus2 at the command line to start MAX+PLUS II.
- Choose **User Libraries** from the Options menu.
- The following libraries should appear in the **Existing Directories** window:

  /local/maxplus2/max2lib/mega_lpm
  /local/maxplus2/max2lib/mf
  /local/maxplus2/max2lib/prim
  /local/maxplus2/max2lib/edif
  /local/maxplus2/vhdl93/altera
  /local/maxplus2/vhdl93/ieee
  /local/maxplus2/vhdl93/lpm

If the above libraries do not appear in the window you must manually search the file system and add them using the **Add** option.

1.2 Overview of MAX+PLUS II

MAX+PLUS II software consists of several application programs and the MAX+PLUS II Manager. The following describes some of the application programs that will be used in this course.

- **Hierarchy Display** - Displays the current hierarchy of files as a hierarchy tree with branches that represent sub-designs. You can tell at a glance which files are currently open and you can also directly open or close one or more files in a hierarchy tree.
- **Text Editor** - The Text Editor lets you create and edit text based logic design files such as VHDL.
- **Compiler** - Processes logic projects targeted for Altera device families (e.g. FLEX 10K). It performs most tasks automatically. However, you can customize all or part of the compilation process.
- **Waveform Editor** - Serves as a tool for entering test vectors and viewing simulation results.
- **Simulator** - Enables you to test the logical operation and internal timing of your logic circuit. Functional simulation, timing simulation, and linked multi-device simulation are available.
- **Timing Analyzer** - Analyzes the performance of your logic circuit after it has been synthesized and optimized by the Compiler.

- **Floorplan Editor** - This editor allows you to view the synthesized circuit, after it has been *partitioned* and *fitted* onto the Field Programable Logic Device (FPLD). (Partitioning and fitting are steps in the compiling pricess). This editor lets you see "what is going on" within the device. You can manually edit physical resource assignments for your project, thereby over-riding the compiler, to imporve your design.

2. Creating a design in VHDL

In this section, you will learn how to enter a VHDL design in text format into the MAX+PLUS II software and to check whether the VHDL code you have entered is correct.

2.1 Example 1: A latch Array

Consider the design of simple array of latches. Figure 1 shows the ***structural*** block diagram of such an entity. It consists of an array of 24 latches, each latch with 24 bits. The VHDL code that describes it structurally is given in Appendix A. Before you continue, please spend some time reading the conventions on writing VHDL code in Appendix C.

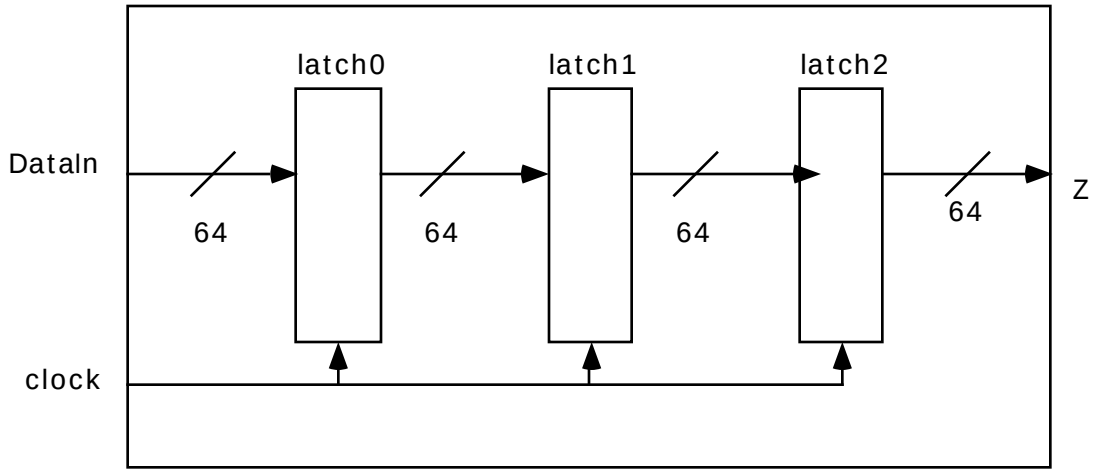


Figure 1 : Structural Diagram of example 1.
Note : The clock signal goes to every flipflop within every latch.

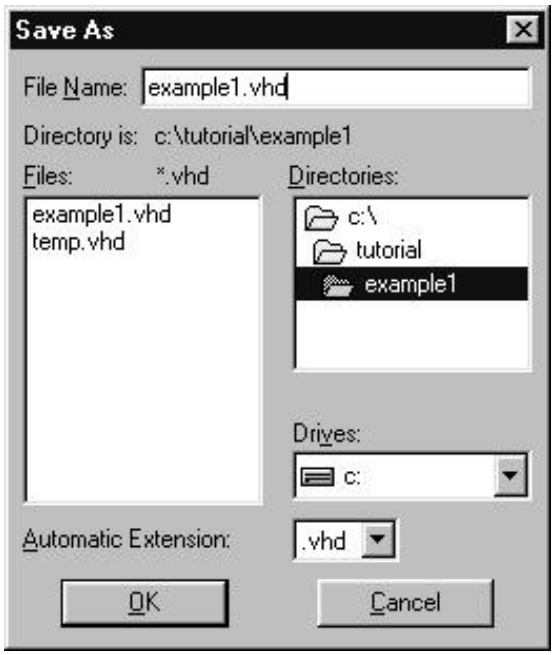
The data are clocked in the left side in parallel 24 bit words. In every clock cycle, a 24 bit word of data moves to the right by one latch. This simple design was selected since it uses all D Flip-Flops in an Altera 10K FPLD. Therefore, the compiler will have a challenge to compile and fit this design. In a real design, you will likely perform a fair bit of combinational logic processing between the stages of latches. Once this design is compiled and placed within the FPLD, the maximum clock rate of this design will serve as the maximum clock rate of any large design with stages of latches, since the addition of combinational logic between the latches will only decrease the speed of this circuit. Therefore, we can probably learn a lot by studying this simple design.

2.2 Entering VHDL code

To enter this design into MAX+PLUS II, follow these steps;

- Choose **NEW** (File menu), select *Text Editor file*, and choose **OK** to open an untitled Text Editor window.
- If necessary maximize the Text Editor window by clicking **Maximize** button on the top right corner of the Text Editor title bar. Type in the VHDL description for example 1 given in Appendix A.
- Choose **Save As** (File menu). Enter example1 into the File Name box and change the Automatic File Extension to .vhd. Click on **OK**. Remember that the name of the file has to be the same as the name of the entity.
- Repeat the steps shown above until you enter all the files in Appendix A. Make sure that you save all VHDL files in the same directory.

Figure 3: Save As dialog box



2.3 Checking your VHDL code

The next step is to check the correctness of your design using the compiler.

- Make sure that you select the window Text Editor - example1. Choose **Project Save & Check** (File menu).
- A dialog box will appear asking whether you want to change the current project name to the file you are going to compile. Click on **YES**. Observe what happens.
- Choose **OK** when the compiler completes its process. If there are no errors and warnings go on to the next subsection.
- If the *Message* window appears, check for error and warning messages. At this point, errors are usually violations of VHDL syntax. Correct the errors and repeat the process.

3. Compiling your VHDL code

3.1 Brief Description of Compiler Stages

The MAX+PLUS II Compiler is a highly automated design processor that transforms design files into output files for device programming, simulation, and timing analysis. When you start the compiler, it begins a series of processes that ultimately creates one or more programming files. While the compiler can compile a project with minimal assistance, it also allows you to customize processing for a particular project. The following are brief descriptions of some compiler stages that you should know. For more details on the compiler stages and their options, please consult the *MAX+PLUS II On-line Help* or the *Altera MAX+PLUS II Getting Started* Manual.

- **Compiler Netlist Extractor.** The compiler first extracts information that defines the hierarchical connections between design files, then checks the overall project for errors.
- **Database Builder**. After the Compiler creates an organizational map of your project, it combines all design files into a fully flattened database for fast and efficient processing.
- **Logic Synthesizer**. The Compiler applies a range of techniques to increase the efficiency of your project and minimize device resource usage. The optional *Design Doctor* utility checks project logic reliability both before and after logic synthesis.
- **Partitioner**. If a project is too large to fit into a single device, the Compiler partitions it among multiple devices from the same device family, either automatically or according to your specifications.
- **Fitter**. The Fitter generates a customizable *Report File* that details resource usage and describes how the project will be implemented in one or more device(s).
- **Timing SNF Extractor**. The optional Timing SNF Extractor creates an *SNF* file which contains the timing data for timing simulation and timing analysis.
- **Functional SNF Extractor**. The optional Functional SNF Extractor creates the functional Simulator Netlist File (.snf) required for functional simulation. The functional SNF does not contain timing information, thus it is generated quicker than the Timing SNF.
- **Assembler**. The Assembler generates one or more files for device programming.

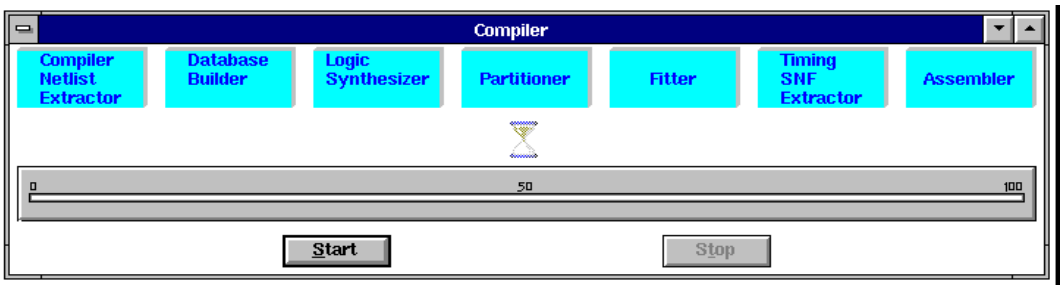


Figure 4: Compiler window

3.2 Setting Compiler Options

In this section, you will learn to change some useful options in the compiler such as *Devices* and *Logic Synthesis* Options in order to achieve your design goal. Introduced here are only frequently-used compiler options. For other options, you must refer to the *MAX+PLUS II On-line Help* or the *Altera MAX+PLUS II Getting Started* Manual.

3.2.1 Selecting a Device Family

You can select any MAX+PLUS II- supported device family for your project. You can also allow the Compiler to automatically choose the most appropriate device within a particular family.

To specify the device family

- Choose **Device** (Assign menu). The **Device** dialog box is displayed.
- Look at the Device Family supported by the MAX+PLUS II by clicking on the arrow at the end of the Device Family box. Select *FLEX10K* if it has not been selected.
- Check on the device in the Devices box. Then turn on the option *Show Only Fastest Speed Grades*, and observe.
- Choose **AUTO** in the list of *Devices* and click **OK**.

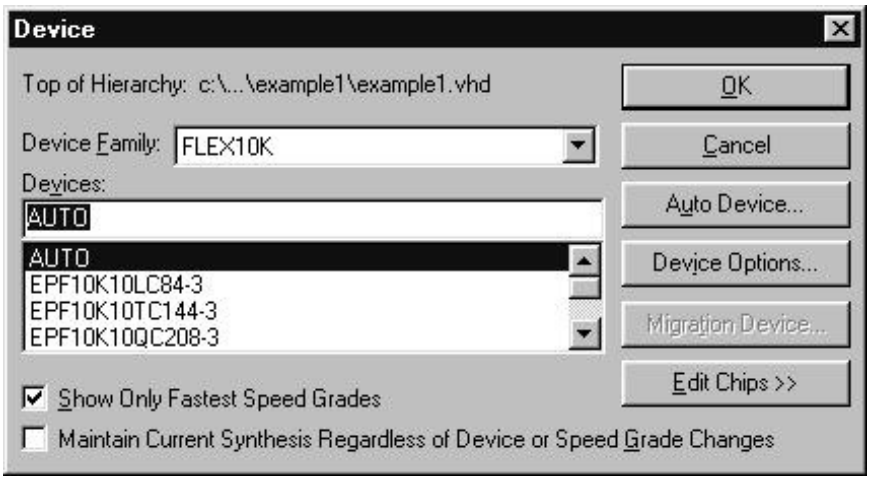


Figure 5: Devices window

3.2.2 Selecting a Global Project Logic Synthesis Style

You can select a logic synthesis style for the project that guides the Compiler's Logic Synthesizer module during compilation. The two main styles of synthesis are *"Minimization of silicon resources"* and *"Minimization of delay"*. You can strive for designs which are hardware efficient but which may be slow, or for designs which are fast but which may be hardware inefficient. The default logic synthesis style for a new project is *"Normal"*, an equal weighting on "speed" and "efficiency".

To select a logic synthesis style for the project.

- Choose **Global Project Logic Synthesis** (Assign menu). The **Global Project Logic Synthesis** dialog box is displayed.

- Move the *Optimize* scroll bar to the middle (5) and click **OK**.

Figure 6: Global Project Logic Synthesis window

- Note the "Automatic Implement EAB" box. You may wish to experiment with this later, to see what it does. For now, leave it unselected. When it is not selected, the complier will implement latches and registers from the pool of programmable D Flipflops within the FPLD. When it is selected, the compiler will likely try to implement latches and memory from the Embedded Array Blocks (EABs) within the FPLD.

3.2.3 Turning on the Smart Recompile Command

When the "smart" recompile feature is turned on, the Compiler saves extra database information for the current project for use in subsequent compilations. During “smart” compilation, the Compiler can determine which modules which do not need to be recompiled, and will skip them during recompilation, thereby reducing compilation time.

To turn on the smart recompile feature:

- Choose **Compiler** (MAX+PLUS II menu). The Compiler window is displayed.
- Choose **Smart Recompile** (Processing menu).

3.2.4 Enabling Compilation for VHDL-93

VHDL is a standard (VHDL-1076) developed by IEEE (Institute of Electrical and Electronics Engineers). It was standardized in 1987 and hence the designation std 1076-1987 or VHDL-87. The standard was revised in 1993 to produce std 1076-1993 or VHDL-93.

By default, MAX+PLUS II will use VHDL-87. To enable compilation for VHDL-93:

- Choose **Compiler** (MAX+PLUS II menu). The Compiler window is displayed.
- Choose **VHDL Netlist Reader Settings** (Interfaces menu).
- Select the **VHDL 1993** button and click OK.

Note that you must repeat these steps for each VHDL file you compile.

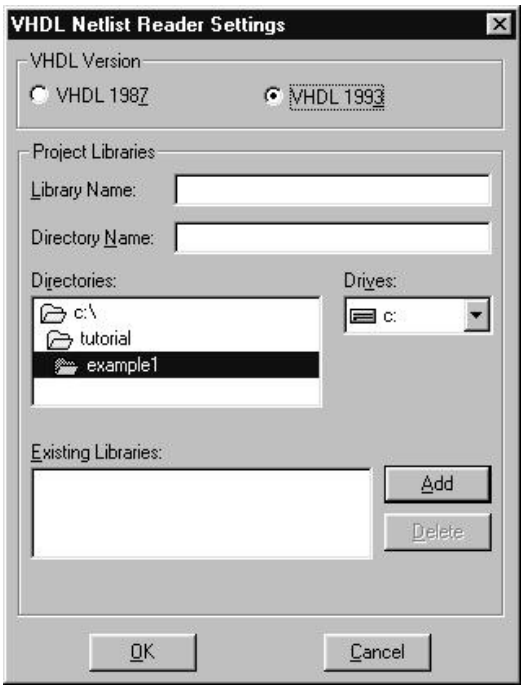


Figure 7: VHDL Netlist Reader Settings dialog box

3.3 Running the Compiler

To compile the project:

- Choose the **Start** button in the Compiler window. As the Compiler processes the project, any information, error, or warning messages appear in a Message Processor window that opens automatically.

4. MAX+PLUS II Simulator

The MAX+PLUS II Simulator provides flexibility and control for simulating single or multi-device projects. The Simulator uses a binary simulation netlist file that is generated during compilation to perform functional, timing, or combined linked multi-device simulation for a project.

You can either define input stimuli two ways: (a) with a straightforward vector input language or (b) you can draw waveforms directly with the MAX+PLUS II **Waveform Editor**. Simulation results can be viewed in the **Waveform Editor** and printed as waveform files.

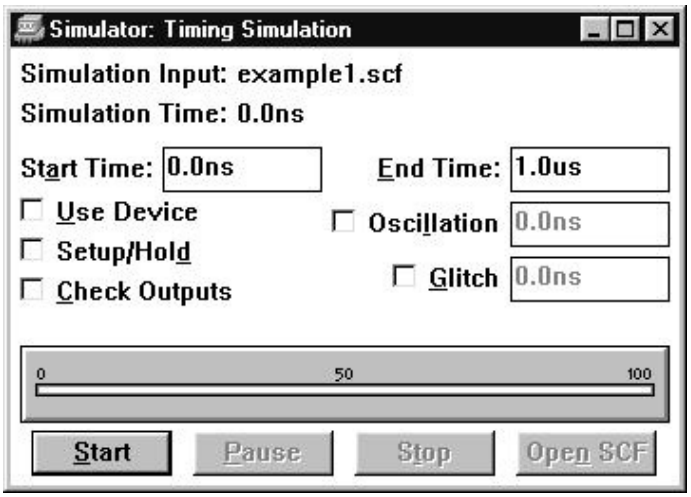


Figure 8: Simulator window

4.1 Functional Simulation versus Timing Simulation

The MAX+PLUS II Simulator supports **functional simulation** to test the logical operation (i.e., functions) of a project **before** it is synthesized, thereby allowing the designer to quickly identify and correct logical errors. Note that in functional simulation mode, gate delays and wire delays are generally ignored; output levels of a logic gate change at the same time as the input levels change. You can turn on functional simulation by choosing **Compiler** (MAX+PLUS II menu) and selecting **Functional SNF Extractor** (Processing menu). Functional simulation is a way to gather fast feedback on the function of your VHDL design, before the design is completely compiled (which requires the steps of synthesis, partitioning and placement to be completed).

In a **timing simulation**, the MAX+PLUS II Simulator tests the project **after** it has been fully synthesized and optimized, partitioned and placed onto the FPLD. Timing simulation considers logic gate delays and wiring delays, and these delays depend upon how the design has been placed onto the FPLD. Timing simulation is performed at 0.1ns resolution. You can turn on timing simulation by choosing **Compiler** (MAX+PLUS II menu) and selecting **Timing SNF Extractor** (Processing menu).

4.2 Timing Simulation using a Simulator Channel File (.SCF)

The **simulator channel file** contains some or all of the nodes in the compiled project. This SCF file is then used to provide inputs and to view the results of the simulation.

To create an SCF file for example 1:

- Make sure that the design of example 1 has been compiled with the Timing SNF Extractor option turned **ON** (see section 4.1).
- Choose **New** (File menu), select *Waveform Editor* file, select the *.scf* extension from the list box and click **OK** to invoke the Waveform Editor with a new, untitled file.
- Choose **End Time** (File menu) and type 2us for 2 microseconds. The end time determines when the simulator will stop applying input vectors. Choose **Grid Size** (Options menu), type 10ns for 10 nanoseconds and click **OK**.
- Choose **Enter Nodes from SNF** (Node menu) to display the dialog box. Turn off the *Group* option under *Type* (The *Inputs* and *Outputs* options should remain turned on). Click **List** to display the available input (I) and output (O) nodes.

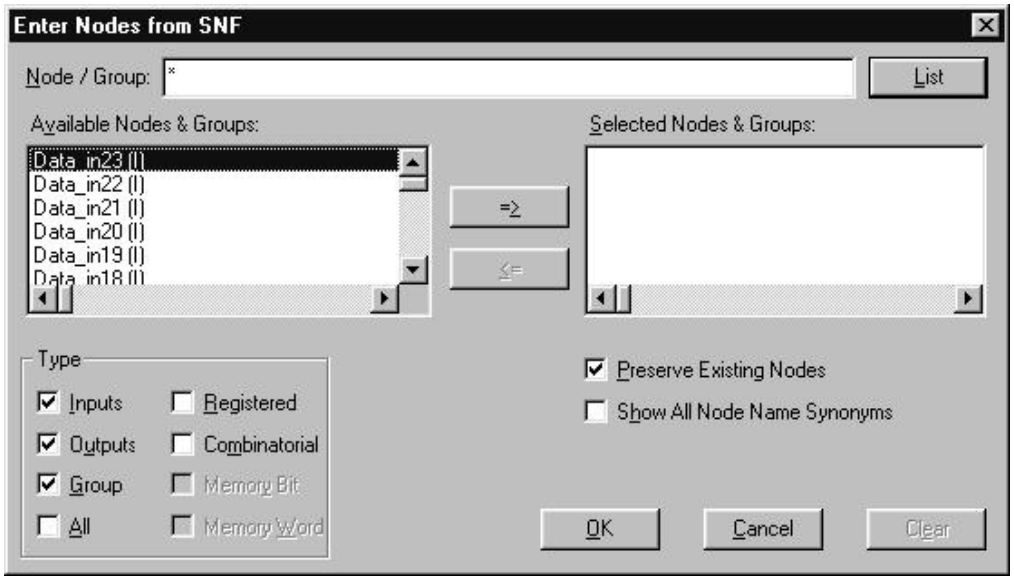


Figure 9: Enter Nodes from SNF window

- Press the mouse button on the topmost node in the *Available Nodes & Groups* box and drag the mouse to highlight all the inputs and outputs. Choose the right direction button (=>) to copy the selected nodes. Alternatively, you may double-click on a node to select it.
- Click **OK**. The selected nodes appear in the Waveform Editor window. All input waveforms have default logic 0. All output waveforms have default undefined (X) states.

- To rearrange the order of signals in the waveform editor window (see fig. 10), press the mouse button on the handle to the left of the node name and move the pointer. A horizontal line displays the current position.
- Choose **Save As** (File menu). The name example1.scf appears automatically in the *File Name* box. Click **OK** to save the file.

4.2.1 Editing a Simulator Channel File

You must edit the input waveforms to provide the input vectors for simulation. As you simulate the project, the simulator overwrites the undefined output waveforms.

To edit the input waveform for the clock:

- Move the pointer and press the mouse button on the Value field for the clock input node. This highlights the entire waveform for this node. Click on the *Overwrite Clock* button from the tool palette on the left side of the window or choose **Overwrite Clock** (Edit menu). To create a clock waveform at the current grid size (10ns), click **OK** to accept the default values. Note that the clock period is actually 20ns.

You can increase/decrease the scale of the waveforms by choosing **Zoom In**/**Zoom Out** (View menu) or clicking on the magnifying glass buttons from the tool palette.

To edit the input waveform for data_in:

- Select the period to overwrite, click the right mouse button, select "overwrite", delete the "groupvalue", and enter a new HEXadecimal value. Repeat this step for every period to be overwritten.

It is posible to observe any other internal signals in the waveform editor window, i.e., reg0_bit0 and reg1_bit0 can be observed as shown in fig. 10.

- Select the period to overwrite, click the right mouse button, select "overwrite", delete the "groupvalue", and enter a new HEXadecimal value. Repeat this step for every period to be overwritten.

We do not have a reset in this design, but here is how you would specify a reset signal if one exists. To edit the input waveform for reset:

- Reset is an active high signal. Use the pointer to highlight the interval from 0ns to 20ns (2 grid units). Click the *Overwrite High (1)* button from the tool palette. Repeat this step to assert a logic high for the interval 0ns to 20ns. To assert logic low, click the *Overwrite Low (0)* button. These commands can also be found in the Edit menu.

Some of the final waveforms are shown below:

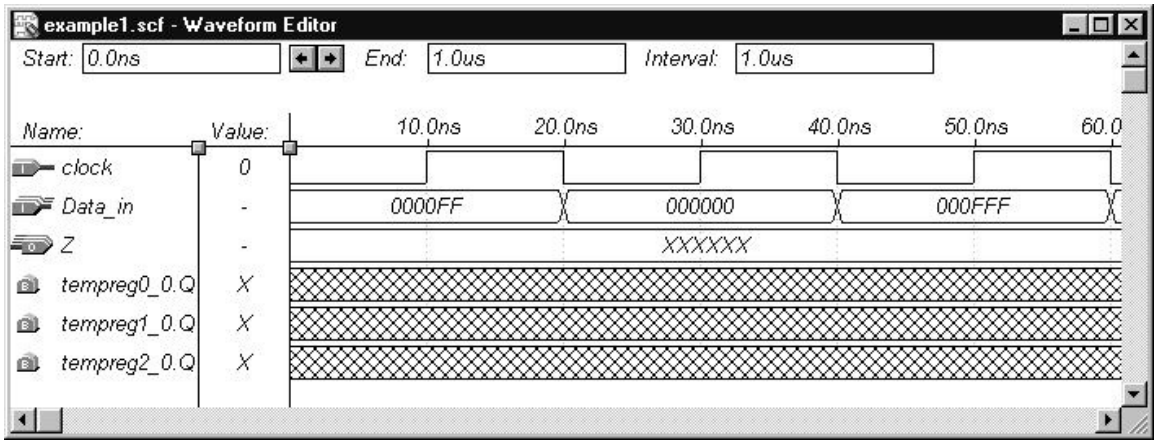


Figure 10: Sample input waveforms

- Save the changes by choosing **Save** (File menu) or simply click on the disk button in the horizontal toolbar.
- Choose **Simulator** (MAX+PLUS II menu).
- Click on *Start* to start the timing simulation.

4.3 Timing Simulation using a Vector File (.VEC)

In this part of the tutorial you will perform a timing simulation of example 1 using a **vector file**. Note that this is an alternative procedure to the one outlined above that uses a Simulator Channel File.

- Make sure that example 1 has been compiled with the Timing SNF Extractor option turned **ON** (see section 4.1).
- Choose **New** (File Menu), select *Text Editor File* and choose **OK** to open an untitled Text Editor window.
- Choose **Save As** (File Menu). Type example1.vec in the *File Name* box. Choose **OK**.
- Type in the vector file as shown in Appendix B of this tutorial.
- Choose **Save** (File Menu).
- Choose **Simulator** (MAX+PLUS II menu).
- Click on *Simulation input*.
- Select example1.vec as the input file and choose **OK**. The simulator will generate an SCF file. Choose **OK** to continue.
- Click on *Start* to start the timing simulation.
- Choose **OK** to continue. If there are any errors or warnings you must go back and check that you entered the vector file correctly.
- Click on *Open SCF* to view the results of the timing simulation.
- Choose **Fit in Window** (View menu) to fit the entire simulation in the Waveform Editor.

5. MAX+PLUS II Timing Analyzer

The MAX+PLUS II Timing Analyzer permits you to analyze the performance of a design after it has been completely compiled (synthesized, optimized, partitioned and fitted onto a FPLD) by the Compiler. You can use the Timing Analyzer to calculate a matrix of point-to-point device delays, determine setup and hold time requirements at device pins, and calculate maximum clock frequency.

5.1 Registered Performance

The Timing Analyzer's Registered Performance Display shows the worst-case registered performance, i.e., the maximum clock frequency for every clock signal in the circuit. The Registered Performance Display measures the maximum delay from the outputs of all flip-flops to the inputs of all other flip-flops, including:

- Clock-to-output delay of the source flip-flop.
- Combinatorial logic and interconnect wiring delays between the source and destination flip-flops.
- Internal setup time of the destination flip-flop.

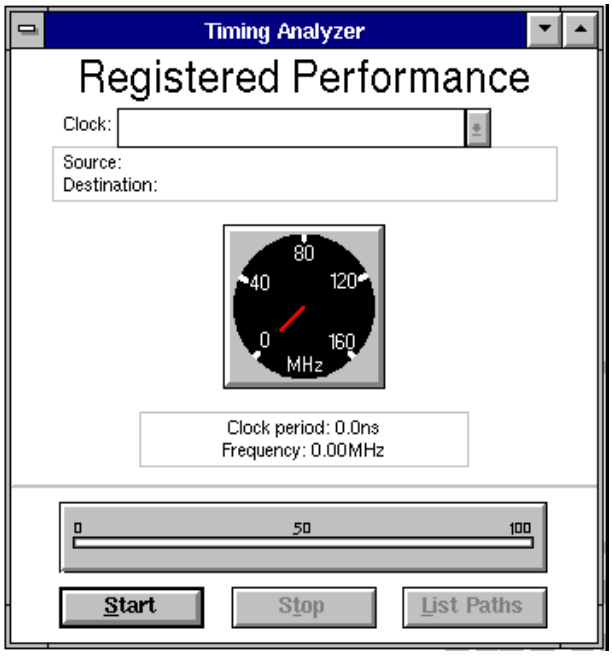


Figure 11: Registered performance window

5.2 Determining the Critical Path

- Choose **Timing Analyzer** (MAX+PLUS II menu).
- Choose **Registered Performance** (Analysis menu).

- Click on *Start* to run the timing analyzer.
- Record the maximum operating frequency of example1 and determine where the longest delay path (critical path) occurs in the circuit.

Note: You can use the options in the **Time Restrictions** dialog box (Options menu) to list either all paths that fail to meet a specified clock frequency or a specified number of paths. After the Timing Analyzer finds the longest delay paths, you can view the information on the paths by choosing **List Paths.** You can choose **Locate** to locate and highlight each signal path in your VHDL file.

Q1: What is the maximum operating frequency of example1?

Q2: Where is the longest delay path (critical path) in the circuit? State briefly where this path lies in the block diagram. Report all of them, if there are more than one path.

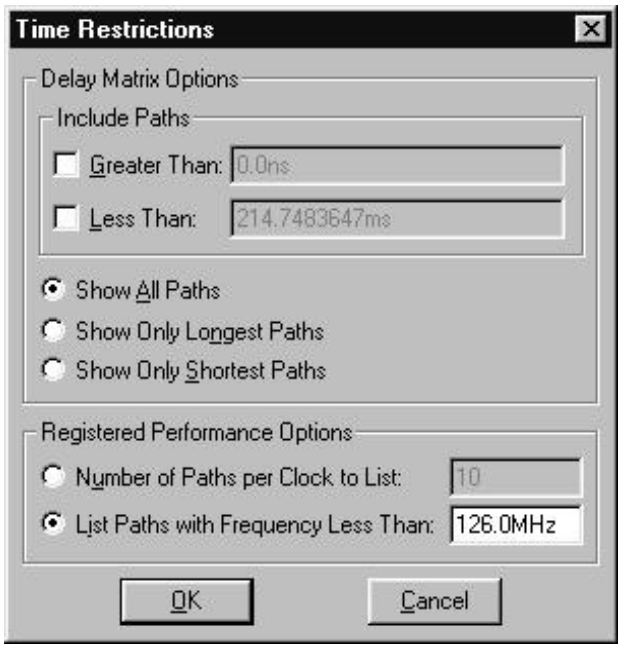


Figure 12: Time restrictions window

5.3 Using the Delay Matrix

The Delay Matrix display shows the *minimum/maximum* propagation delays (combinational logic and wiring delays only) between source input pins and destination output pins which are not connected via registers in the current project. The delay matrix is useful when your FPLD device implements many combinational logic paths without internal latches. The delays between registers are reported in fig. 12.

A single delay time means all combinational delay paths are of the same length. To exclude the minimum or maximum delays from the display, and to restrict the lengths of the delay paths used for the analysis, follow the notes in section 5.2 about setting the options in the **Time Restrictions** (Option menu).

- Choose **Timing Analyzer** (MAX+PLUS II menu).
- Choose **Delay Matrix** (Analysis menu).
- Click on *Start* to run the timing analyzer.
- In our case, there are no purely combinational logic paths between input pins and output pins, so the delay matrix is empty. However, if your FPLD was in a system where it was replacing many discrete logic chips by implkementing combinational "glue" logic, the delay matrix would be very helpful to track the delays of every combinational logic pathway.

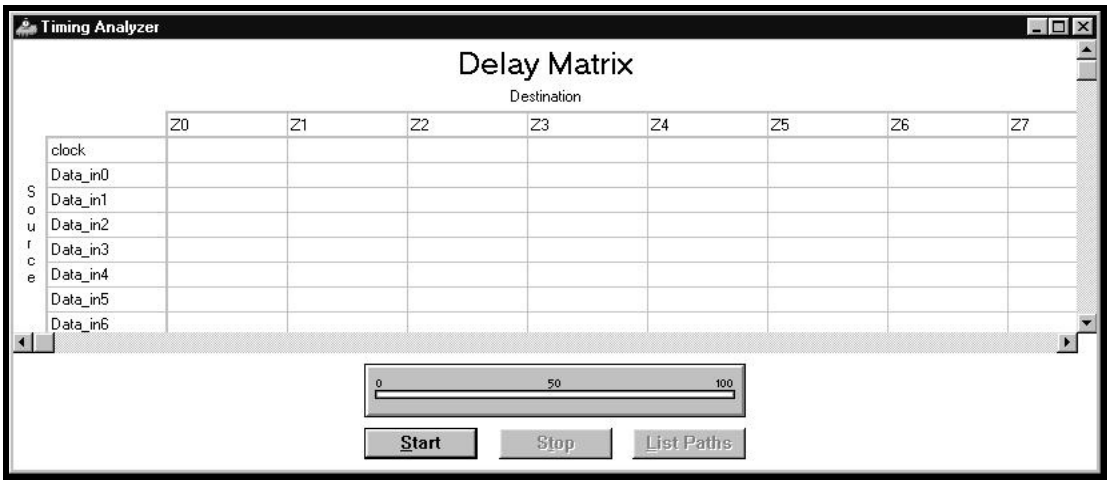
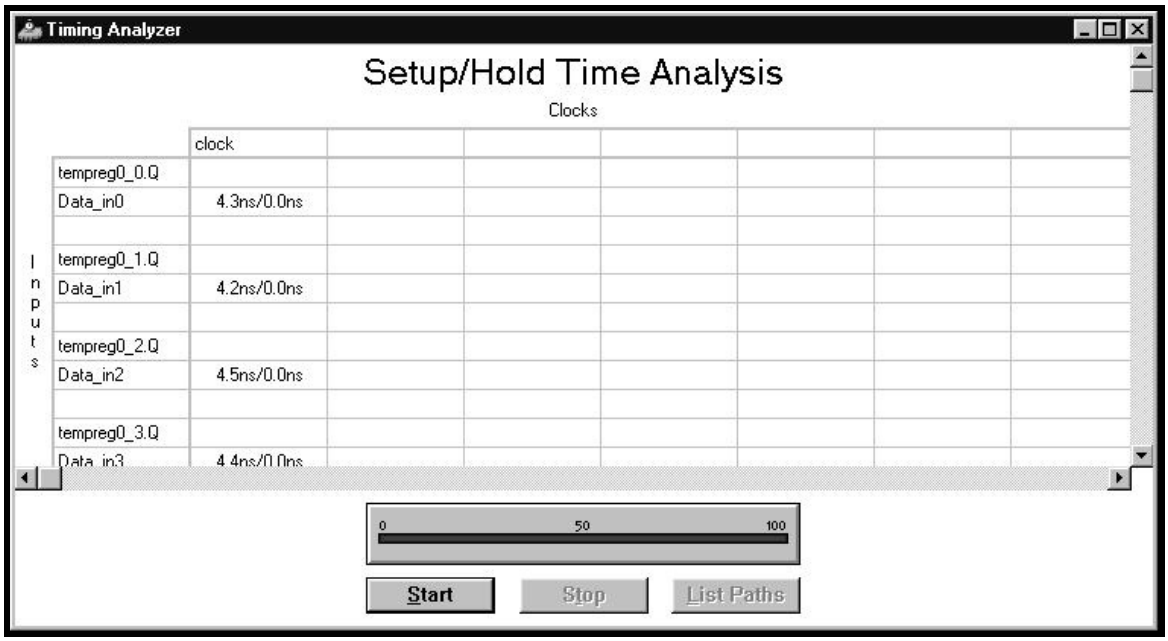


Figure 13: Delay matrix display

There is a related window called the **setup and hold matrix**, which appears as follows. This window reports the setup and hold time margins between at every DFF in your design, with respect to the clock. You can examine any DFF and see how much spare time (if any) you have in the combinational logic path leading to the inputs of that DFF. If you have plenty of spare time, you can add more combinational logic to the path to use up the spare time, without changing the maximum clock rate. If you have no spare time, you cannot add more logic to that path without slowing down the clock, i.e., that path is the critical path which specifies the overal clock rate. This window is useful when you are designing a very high performance pipelined circuit, and you want to find out where you have some spare time to perform logical processing between pipeline stages.



6. Analyzing Synthesis, Partitioning and Fitting Results (.RPT file)

The **report file** is a text file which contains useful information on how a design is implemented in a target programmable logic device. To view the .RPT file, click on the RPT icon corresponding to the entity in the hierarchy tree of the current project. Alternatively, you may double-click on the report file icon in the compiler window.

Information on the project as a whole (e.g. name of the target device, no. of user I/O pins, percentage of the resources used in the device) and project compilation messages are listed first.

Next, the device specific information follows. A pin-out diagram of the target device shows the mapping of user I/O, VCC, GND and special purpose signals to the physical pins on the chip. The resource usage section provides a concise description of how the device uses available resources. The fan-in and fan-out statistics of all input, output, buried (internal), clock and clear signals are listed.

At the end of the .RPT file is a summary of the compilation and logic synthesis settings used for the current design.

Compile your design two ways. First, use the NORM setting and view the layout in the floorplan editor. Secondm using ther FAST setting and view the layout. Answer the following question for each result.

The NORM setting causes the compiler to place adders in your design into the pool of programmable Logic Blocks, where the rinternal ipple-carry chains in the adders are symthesized using programmable logic gates.

The FAST setting causes the compiler to place adders in your design into the pool of programmable Logic Blocks, in a special manner which exploits built-in "fast-carry-chain" hardware within the Logic Blocks. If your design uses many adders, then compiling it with the FAST setting will result in faster and more

hardware efficient addrers, since the ripple-carry chains will be realized with specialized hardwired logic gates within the Logic Block, rather than by using programmable logic gates.

Q: What is the percentage of resources (e.g., Logic Blocks, Logic Elements, I/O pins) that are used for example 1? Include only appropriate sections of the report file.

6. MAX+ PLUS II Floorplan Editor

The **floorplan editor** illustrates how your design has been assigned to the resources of the FPLD. Altera 10K FPLDs have a pool of programmable Logic Blocks (LBs), and Embedded Array Blocks (EABs). These are described elsewhere. You can see how your design has been mapped onto the FPLD using this editor. Try this editor, using both designs (compiled with the NORM and FAST settings).

The MAX+PLUS II floorplan editor allows you to view Compiler partitioning and fitting results, as well as to enter and edit physical device resource assignments for your project. You will also view the Compiler's assignments, and back-annotate the results of compilation.

6.1 Open the Floorplan Editor

The floorplan editor provides two displays - the Device View and the LAB View. The Device View shows all pins on a device package and their function. The LAB View shows the interior of the device, including all LABs; the individual logic cells within each LAB; and I/O cells, embedded cells, and Embedded Array Blocks (EABs). It also displays pin locations so that you can see the relationships between pins and logic resources in the interior of the device.

- Select **MAX+PLUS II | Floorplan Editor** to open the window shown in Figure 14.
- Select **Layout** in the menu or Double Click on the device to display **Device View** and **LAB view**.
- Use the scroll bars, "positive" and "negative" magnify buttons to change the view of the device.

6.2 Display Routing Information

The floorplan Editor allows you to view the routing information for one or more selected logic cells, pins, and assignment bins using a variety of different methods. You can also view routing statistics for any part of the current chip.

6.2.1 Display node fan-in and fan-out routing information

The Floor plan Editor shows the fan-in (pink) and fan-out (blue) routing lines that apple to the selected items. Fan-in and fan-out lines are updated automatically when you move an assignment.

- Turn on **Show Node Fan-in** or **Show Node Fan-Out** from options menu.

OR

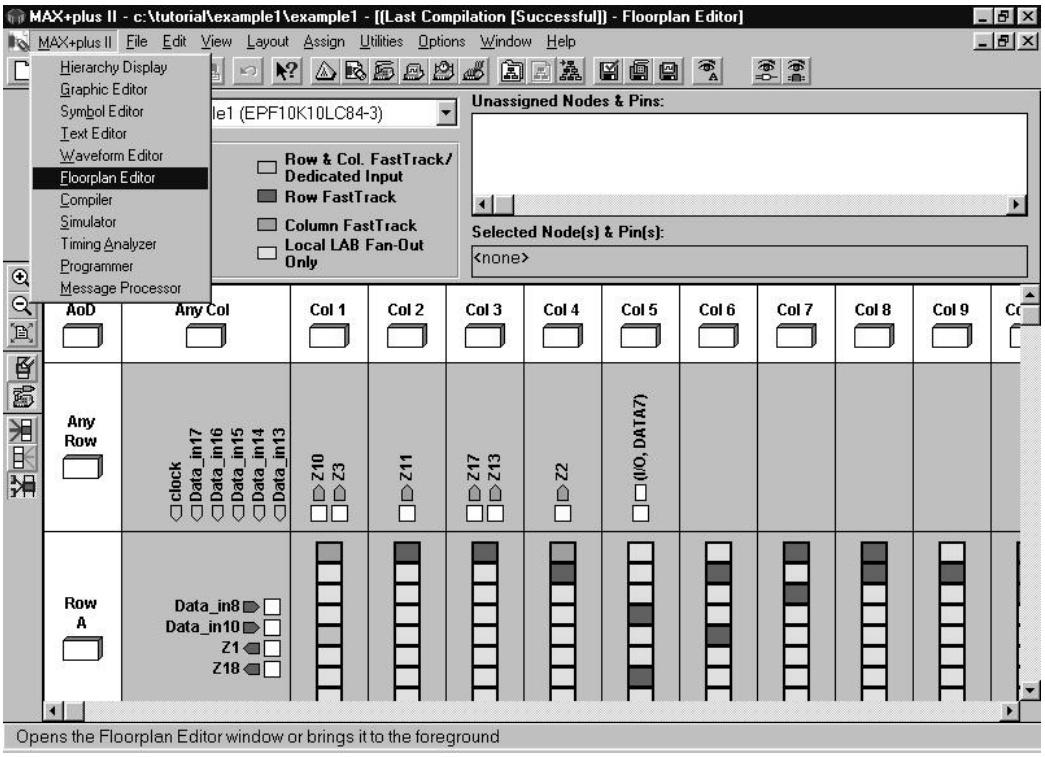


Fig. 14. Floorplan Editor

· Choose the Show Node Fan-in and Show Node Fan-Out buttons on the tool palette. (left hand side)

· Select one or more logic cells, pins or assignment bins in the **LAB View**.

6.2.2 Display Signal Path

You can also choose to view only the signal path between two or more items, without additional fan-in and fan-out information:

· Turn on **Show Path** in the **Options** menu.

Or

· Choose the **Show Path** button on the tool palette.

This will allow you to view only the connection between the selected nodes, and is especially useful for tracing critical timing paths.

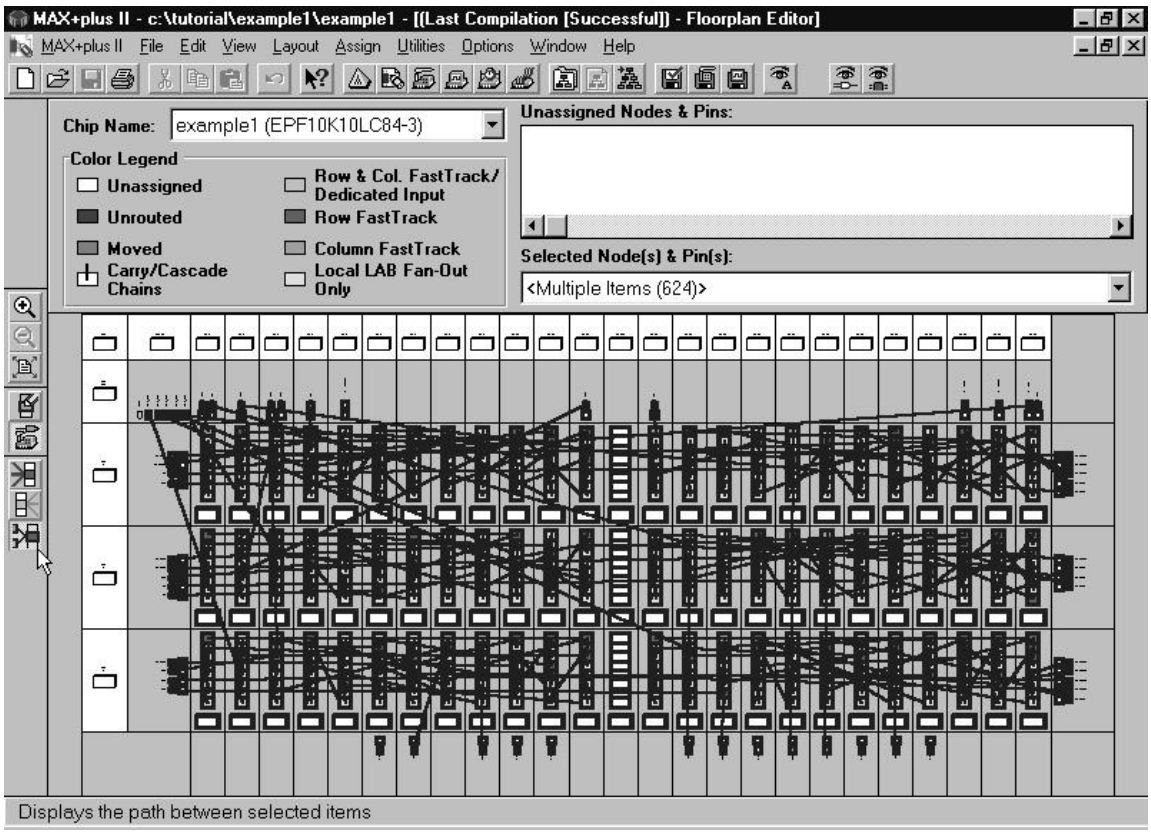

MAX+plus II - c:\tutorial\example1\example1 - [(Last Compilation [Successful]) - Floorplan Editor]
MAX+plus II File Edit View Layout Assign Utilities Options Window Help
Chip Name: example1 (EPF10K10LC84-3)
Color Legend
Unassigned
Unrouted
Moved
Carry/Cascade Chains
Row & Col. FastTrack/ Dedicated Input
Row FastTrack
Column FastTrack
Local LAB Fan-Out Only
Unassigned Nodes & Pins:
Selected Node(s) & Pin(s):
<Multiple Items (624)>
Displays the path between selected items

Figure 15: Signal Path

6.2.3 Display Routing Statistics

· Select one or more logic cells, pins or assignment bins.

· Choose **Routing Statistics** in **Options** menu

OR

· Double click on a single item

· Choose the **Calculate Most Congested Areas** button.

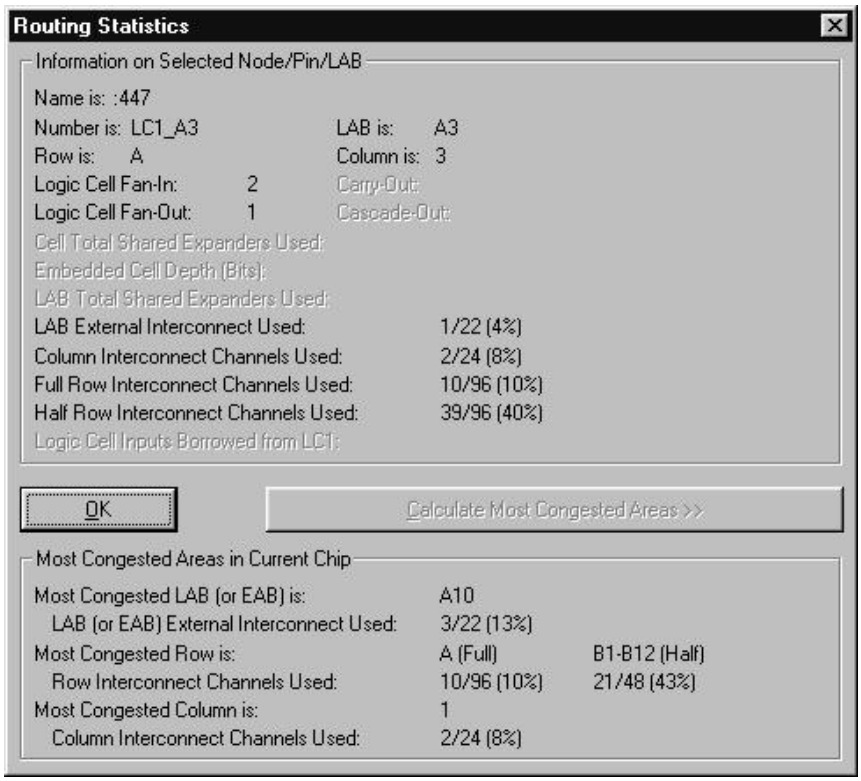

Routing Statistics
Information on Selected Node/Pin/LAB
Name is: :447
Number is: LC1_A3
LAB is: A3
Row is: A
Column is: 3
Logic Cell Fan-In: 2
Carry-Out:
Logic Cell Fan-Out: 1
Cascade-Out:
Cell Total Shared Expanders Used:
Embedded Cell Depth (Bits):
LAB Total Shared Expanders Used:
LAB External Interconnect Used: 1/22 (4%)
Column Interconnect Channels Used: 2/24 (8%)
Full Row Interconnect Channels Used: 10/96 (10%)
Half Row Interconnect Channels Used: 39/96 (40%)
Logic Cell Inputs Borrowed from LC1:
OK
Calculate Most Congested Areas >>
Most Congested Areas in Current Chip
Most Congested LAB (or EAB) is: A10
LAB (or EAB) External Interconnect Used: 3/22 (13%)
Most Congested Row is: A (Full) B1-B12 (Half)
Row Interconnect Channels Used: 10/96 (10%) 21/48 (43%)
Most Congested Column is: 1
Column Interconnect Channels Used: 2/24 (8%)

Figure 16: Routing Statistics Window

6.3 Back-Annotate the Project & Edit Assignments

The Floorplan Editor allows you to view and edit your current assignments, which are stored in the project's Assignment & Configuration File (.acf). After you have compiled the project, you can edit the Compiler's assignments, which are stored in the project's Fit File (.fit).

- Choose **Back-Annotate** Project from the **Assign** menu.
- Turn on the *Chips, Logic Cells, Pins & Devices* option
- Choose **OK**.
- Choose **Current Assignment Floorplan** in the **Layout** menu. (or choose the equivalent button)

Figure 17: Back-Annotate Option Window

Note: The Floorplan Editor cannot display a floorplan for a chip assigned to an AUTO device.

MAX + Plus II copies the pin logic cell, chip and device assignments from the Fit file into the ACF and display the current assignments for the project. The Floorplan Editor provides a list of unassigned node and pin names if the logic is unassigned in your project. You can now use the "drag and drop" method to move a node or pin to a different location on the device.

You can turn on **Show Moved Nodes** in Gray in the **Options** menu, so that you can easily track the differences between the current assignments and the last compilation fit.

Once you have edited the pin/node assignments, you must recompile your project to verify whether or not your new assignment is legal for the currently selected device.

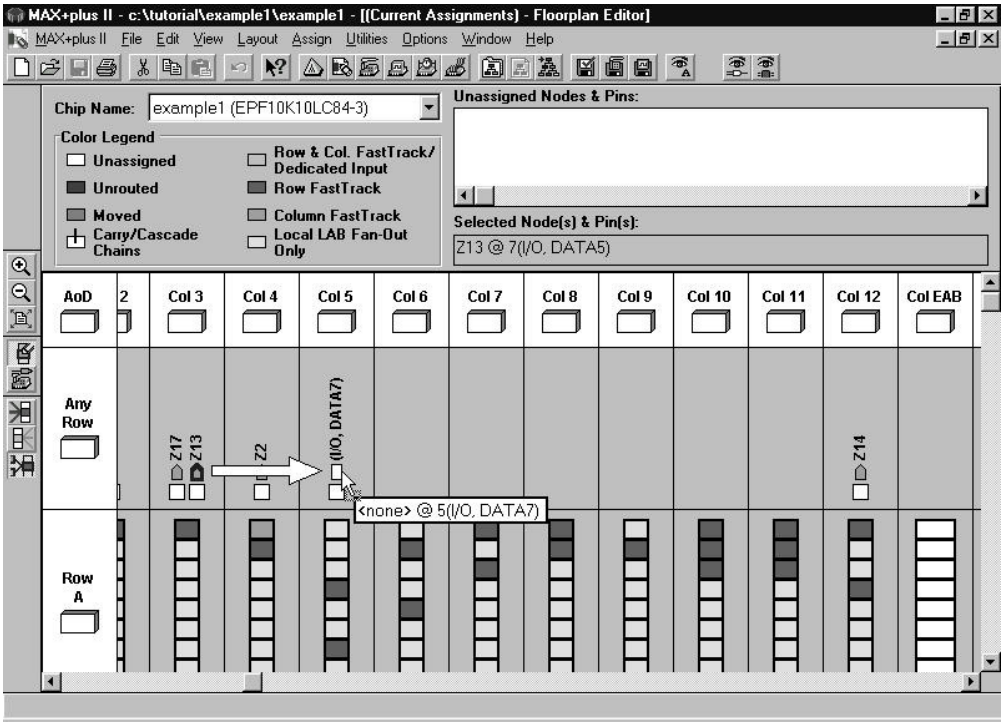


Figure 18: Drag and Drop to Edit Pin/Node

7. Changing Compiler Settings

7.1 Viewing logic synthesis equations

Optional sections can be included in the report file. A more detailed report file can help you locate errors in a project that does not compile successfully.

- Invoke the Compiler and choose **Report File Settings** (Processing menu). Turn on the *Equations* option to include the synthesized logic equations in the report file.
- Recompile the project and view the report file.

The Equations section provides the results of extensive logic synthesis. Since synthesis minimizes the logic required to implement a design, redundant or unnecessary logic in the original design may not appear in the report file.

Hint: Since the *Equations* section in the report file can be very lengthy, it is generally sufficient to turn on only the *User Assignments* and *File Hierarchy* options in the **Report File Settings**.

7.2 Optimizing logic synthesis for speed

Hardware designers are often faced with the speed-area trade-off. Here, we briefly look at the effect of optimizing the synthesis of the design for speed.

- Choose **Global Project Logic Synthesis** (Assign Menu) to display its dialog box.
- Under *Optimize*, move the scroll bar towards Speed until it reads 10. Click OK to confirm changes.
- Recompile the project.

Q7: What are the effects of this optimization on the timing performance (registered performance and delay matrix) and resource usage? Give a brief explanation on your answer.

8. Guidelines

Here are some guidelines that might be useful while using the MAX+PLUS II development system.

- You are encouranged to use the resources on the EE4DM4 Computer Architecture Laboratory course web page. The URL is on the front page of this manual.
- MAX+PLUS II on-line help contains all of the MAX+PLUS II documentation that comes with the development system. Therefore, you should consult the on-line help whenever you encounter a problem.
- Use only lower case characters for all filenames.
- The entity name in your VHDL description must be the same as the filename.
- Do not label any signal I/O port with the same name as an entity.
- Use the Compiler's **Design Doctor** utility to check the reliability of your design against one or more selected design rules.
- Introduction to the toolbar shortcuts:

The toolbar is located along the top of the MAX+PLUS II window. When you put the mouse pointer over any button in the toolbar, a one-line description of the button function will appear at the lower-left corner of the MAX+PLUS II window. Clicking on toolbar buttons allow you to quickly access various MAX+PLUS II applications and options.

Button 1: Opens the **Hierachy Display** window.

Button 2: Opens the **Floorplan Editor** window, which allows you to view and edit pin and logic cell assignments for the current design.

Button 3: Opens the **Compiler** window.

Button 4: Opens the **Simulator** window.

Button 5: Opens the **Timing Analyzer** window.

Button 6: Opens the **Programmer** window, which together with the appropriate hardware allows you to program Altera devices.

Button 7: Changes the name of the current project.

Button 8: Sets the current file to be the current project.

Button 9: Opens the top-level file of the current project.

- You may use the Altera VHDL library to implement *standard functions* such as counters, multiplexors, shift registers etc. Select the **Old-Style Macrofunctions** option (Help menu) to get more details. All VHDL component declarations of the standard library can be found in:

  /local/maxplus2/vhdl93/altera/maxplus2.vhd.

  To reduce development time, you should use MAX+PLUS II's Library of Parameterizable Modules (also called **LPM** functions). By specifying a set of parameters, known as *generics*, you can customize these modules to the specification or functionality of the components in your design (e.g. the no. of I/O ports, the width of each port). For more details, select the **Megafunctions / LPM** option (Help menu).

- MAX+PLUS II only supports a *subset* of the VHDL language. Therefore, you must check whether a certain feature is supported by MAX+PLUS II before you can use it. You can do this by selecting the **VHDL** option (Help menu).

- MAX+PLUS II provides *VHDL templates* as an easy and accurate way for you to enter VHDL syntax. Once you have inserted a template into VHDL file, you must replace all variables with your own logic. Each variable name start with double underscore (__) and each keyword is capitalized. To insert a VHDL construct, position your cursor at the desired location in your VHDL file. Select **VHDL Template** (Templates menu) to bring up the VHDL Template dialog box. Then choose the desired construct from the list.

- Syntax Coloring highlights comments, keywords, identifiers etc. in different colours on the screen. Hence, it may be useful when editing and debugging your code. To enable this feature, select the **Syntax Coloring** option from the Options menu.

9. Resources

- Altera Data Book 1996
- Altera MAX+PLUS II Getting Started
- Altera MAX+PLUS II VHDL
- Altera MAX+PLUS II On-line Help
- "A VHDL Primer", Revised Edition, by Jayaram Bharsker, 1995.
- "The Designer's Guide to VHDL", by Peter J. Ashenden, Morgan-Kaufman Publishers, 1995.
- "Digital Systems Design and Prototyping Using Field Programmable Logic", by Zoran Salcic and Asim Smailagic, Kluwer Academic Publishers, 1997.
- "VHDL and FPLDs in Digital Systems Design, Prototyping and Customization", by Zoran Salcic, Kluwer Academic Publishers, 1998.
- Computer Architecture class home page
- Altera home page (www.altera.com)

Appendix A - VHDL Description

```
-- This package is for Example 1 only


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;  -- definitions for standard logic vectors


PACKAGE temp IS

      constant max_latch: integer := 2;
      constant max_bit: integer := 63;
      SUBTYPE latch_type is STD_LOGIC_VECTOR(max_bit downto 0);
      TYPE latches IS ARRAY (max_latch DOWNTO 0) OF latch_type;


END temp;
```

```
--------------------------------------------------------------------------
-- Example 1: array of 24 Latches, each with 24 bits
--------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;        -- definitions for standard logic vectors
USE IEEE.STD_LOGIC_SIGNED.ALL;      -- definitions for signed arithmetic
USE work.temp.all;


-- Entity declaration

ENTITY latch3 IS
      PORT(Data_in      :IN std_logic_vector(max_bit downto 0);

            Z           :OUT std_logic_vector(max_bit downto 0);

            clock       :IN STD_LOGIC

            );

END lacth3;


- Architecture declaration

ARCHITECTURE latch3_arch OF latch3  IS

SIGNAL  reg       : latches;

-- Note : Signals values which are assigned within a process take effect only upon
-- exit of the process. Latch values which are assigned within a process take effect
-- as soon as they as assigned.

BEGIN

Input: PROCESS (clock)  -- call this process "input"

      BEGIN
                  IF(clock'EVENT AND clock='1') THEN

                  reg(0) <= Data_in;  -- will overwrite reg(0) on exit of process

                  for i in 0 to (max_latch-1) loop

                        reg(i+1) <= reg(i);  -- shift contents up on exit of process

                  end loop;

                  Z <= reg(max_latch);   -- copy last register to output port

            END IF;

      END PROCESS;
                                    END latch3_arch;
```

Appendix B - Vector File: example1.vec

```
% Units default to ns. Input values are interpreted in hexadecimal format by default. %

% Specify start and stop time of simulation. %

START 0;
STOP 600;

% Clock ticks every 10 ns i.e., clock period = 20 ns. %

INTERVAL 10;
INPUTS clock;
PATTERN
0 1;                                % Relative vector values %


INPUTS data_in;
PATTERN
0>              FFFF0000FFFF0000
20>             0000000000000000
40>             FFFF0000FFFF0000
60>             0000000000000000
60>             FFFF0000FFFF0000            % input constant hereafter
;


% select Output traces  to view in waveform editor %

OUTPUTS Z;
```

Appendix C - EE-4DM4 Class Conventions on writing VHDL documentation

In a large project with multiple contributors it is important to have well documented professional code. The code should be easy to locate, understand and maintain by a person other than the original author. The code should facilitate "re-use", i.e., it should be structured into modules which can be reused by other members in other parts of the project. The following conventions are expected in your assignments and projects; this will help the TAs and instructors to assess your code.

Convention 1: Each VHDL file should start with a consistent professional-style **header**.

Convention 2: The **entity** and corresponding **architecture** description should be kept in the same file.

Convention 3: The name of the file should match the **entity** name with a **.vhd** suffix to facilitate easy location and maintenance.

Convention 4: The name of the architecture should include the **modelling style (struct or arch)**. If the entity name is "*latch1*" and the architectural modelling style is "*structural*", then a suitable architecture name would be "*latch1_struct*".

Convention 5: The header for an entity with a complex structural description should refer the reader to a **structural diagram** in the lab. report, unless the entity is trivial. The structural circuit diagram should conform to the usual circuit diagram conventions, i.e. illustrate *every internal component* and *every I/Oport of the entity.* An entity with a simple behavioral description may not have a structural diagram.